Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Dr. Peter Lammich

SS 2017
21. 7. 2017

Functional Data Structures

Exercise Sheet 13

Presentation of Mini-Projects

You are invited, on a voluntary basis, to give a short presentation of your mini-projects in the tutorial on July 28.

Depending on how many presentations we have, the time slots will be 5 to 10 minutes, plus 2 minutes for questions.

If you are interested, please write me a short email until Thursday, July 27.

Exercise 13.1 Double-Ended Queues

Design a double-ended queue where all operations have constant-time amortized complexity. Prove functional correctness and constant-time amortized complexity.

For your proofs, it is enough to count the number of newly allocated list cells. You may assume that operations *rev xs* and *xs* @ *ys* allocate $O(|xs|)$ cells.

Explanation: A double-ended queue is like a queue with two further operations: Function *enq_front* adds an element at the front (whereas *enq* adds an element at the back). Function *deq_back* removes an element at the back (whereas *deq* removes an element at the front). Here is a formal specification where the double ended queue is just a list:

abbreviation (*input*) *enq_list* :: *"'a ⇒ 'a list ⇒ 'a list"* **where**
"enq_list x xs ≡ xs @ [x]"

abbreviation (*input*) *enq_front_list* :: *"'a ⇒ 'a list ⇒ 'a list"* **where**
"enq_front_list x xs ≡ x # xs"

abbreviation (*input*) *deq_list* :: *"'a list ⇒ 'a list"* **where**
"deq_list xs ≡ tl xs"

abbreviation (*input*) *deq_back_list* :: *"'a list ⇒ 'a list"* **where**
"deq_back_list xs ≡ butlast xs"

Hint: You may want to start with the *Queue* implementation in *Thys/Amortized_Examples*.

lemma *"list_of init = []"*
lemma *"list_of(enq x q) = enq_list x (list_of q)"*
lemma *"list_of(enq_front x q) = enq_front_list x (list_of q)"*

lemma *"list_of q ≠ [] ⟹ list_of(deq q) = deq_list (list_of q)"*
lemma *"list_of q ≠ [] ⟹ list_of(deq_back q) = deq_back_list (list_of q)"*

Homework 13 Pairing Heap

Submission until Friday, 28. 07. 2017, 11:59am.

The datatype of pairing heaps defined in the theory `Thys/Pairing_Heap` comes with the unstated invariant that *Empty* occurs only at the root. We can avoid this invariant by a slightly different representation:

datatype *'a hp = Hp 'a "'a hp list"*
type_synonym *'a heap = "'a hp option"*

Carry out the development with this new representation. Restrict yourself to the *get_min* and *delete_min* operations. That is, define the following functions (and any auxiliary function required)

fun *get_min* :: *"('a :: linorder) heap ⇒ 'a"* **where**
fun *del_min* :: *"'a :: linorder heap ⇒ 'a heap"* **where**
fun *php* :: *"('a :: linorder) hp ⇒ bool"* **where**
fun *mset_hp* :: *"'a hp ⇒'a multiset"* **where**
fun *mset_heap* :: *"'a heap ⇒'a multiset"* **where**

and prove the following functional correctness theorems and any lemmas required, but excluding preservation of the invariant:

theorem *get_min_in*: *"get_min (Some h) ∈ set_hp(h)"*
lemma *get_min_min*: *"⟦ php h; x ∈ set_hp(h) ⟧ ⟹ get_min (Some h) ≤ x"*
lemma *mset_del_min*: *"mset_heap (del_min (Some h)) = mset_hp h − {#get_min(Some h)#}"*

It is recommended to start with the original theory and modify it as much as needed.

Note that function *set_hp* is defined automatically by the definition of type *hp*.